CS422: Computer Architecture
Assignment 3

Sharvil Sachin Athaley (210961)
Samyak Singhania (210917)
Sanath Salampuria (210919)

April 21, 2025

Part I: Instruction Mix

Test Case	**Total Instr.**	**Loads (%)**	**Stores (%)**	**Branches (%)**
asm-sim	161	39.13	34.7	0.00
c-sim	1931	17.45	8.23	19.26
endian	1968	19.15	11.07	17.98
factorial	2533	16.50	8.64	18.39
fib	44906	13.60	17.77	8.34
hello	1747	19.37	9.03	19.57
host	14897	20.10	9.92	19.21
ifactorial	2423	16.50	7.88	19.23
ifib	7584	16.99	10.02	17.49
log2	2099	17.91	8.38	19.58
msort	17222	10.59	5.44	18.12
rfib	25671	14.40	17.03	9.51
subreg	3721	17.52	8.30	19.18
towers	82766	20.05	8.48	20.54
vadd	7262	6.15	16.35	10.57

It is interesting to note that ifib and ifactorial have much lesser instructions than rfib and factorial as they are iterative.

Part II: CPI for All Designs

The following five processor designs were evaluated in terms of their performance, specifically by measuring their average CPI (Cycles Per Instruction) across various benchmark programs:

- **Unpipelined:** Unpipelined processor (Note CPI is 5 here for all programs).
- **Design-1:** Interlocked pipeline with no bypass paths.
- **Design-2:** Improved version of Design-1 with a single branch delay slot.
- **Design-3:** Added EX-EX bypass path to Design-2.
- **Design-4:** Added MEM-MEM bypass path to Design-3.
- **Final Design:** Includes all bypass paths (EX-EX, MEM-MEM, MEM-EX) with a load delay slot and branch delay slot (filled by compiler).

The table below presents the CPI for each design across all test cases:

TestCase	Design-1	Design-2	Design-3	Design-4	Final Design	Load delay stalls (%)
asm-sim	1.64	1.51	1.16	1.16	1.16	0
c-sim	2.19	2.04	1.89	1.88	1.01	0
endian	2.09	1.95	1.15	1.15	1.01	0
factorial	2.18	2.02	1.12	1.11	1.02	0
fib	1.90	1.71	1.17	1.16	1.00	0
hello	2.15	2.01	1.17	1.17	1.01	0
host	1.94	1.78	1.19	1.19	1.00	0
ifactorial	2.19	2.04	1.19	1.19	1.01	0
ifib	2.04	1.90	1.09	1.09	1.00	0
log2	2.11	1.96	1.13	1.13	1.01	0
msort	2.25	2.10	1.08	1.08	1.00	0
rfib	1.91	1.73	1.09	1.08	1.00	0
subreg	2.14	1.99	1.12	1.11	1.01	0.08
towers	1.92	1.76	1.15	1.15	1.00	0
vadd	2.45	2.04	1.09	1.09	1.00	0

Approach and Analysis

We have implemented the final design (with bypass paths and load delay slot) in the following manner :

1. We maintained a a buffer gpr array in which values were updated as soon as they were computed (i.e. in EX for ALU instruction and MEM for loads). These buffer values were then used by instructions for computations in the EX stage. These were our 'bypass paths'.

2. The above bypass paths along with phased write-read (i.e. WB writing on positive edge and ID reading on negative edge) essentially eliminates data hazards.

3. Load delay slot i.e. hazards when the previous instruction is a load still remain. They are detected and handled by interlocking in the decode stage.

4. Design 1 with stalls for load delay, data hazard and branch had also similarly detected above problems and stalled in the decode stage.

5. The reason that some of the programs still have $CPI > 1$ may be that syscalls take multiple cycles to execute and this increases number of cycles even though only 1 instruction is being executed (the syscall itself) . Note that load delay slot is not the reason as load delays were found to be 0 for all programs except subreg.c.

Some more statistics:

Test Case	Instructions	Syscalls	Percentage Syscalls (%)
asm-sim	161	3	1.84
c-sim	1931	5	0.25
endian	1968	5	0.25
factorial	2533	5	0.19
fib	44906	5	0.01
hello	1747	5	0.28
host	14897	24	0.16
ifactorial	2423	5	0.20
ifib	7584	5	0.06
log2	2099	5	0.23
msort	17222	5	0.02
rfib	25671	5	0.01
subreg	3721	7	0.18
towers	82766	67	0.08
vadd	7262	5	0.06

We also recorded load delay stalls and found them to be 0 in the original 14 programs and 3 in number in subreg.c.